VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Použití knihovny LibUSB v prostředí WIN32 pro komunikaci s mikropočítačem přes USB rozhraní.

Communication with Microcontroller over USB Using Library LibUSB

2013 Lukáš Patočka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Patočka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Použití knihovny LibUSB v prostředí WIN32 pro komunikaci s mikropočítačem přes USB rozhraní
Communication with Microcontroller over USB Using Library LibUSB

Zásady pro vypracování:

Navrhněte a vyzkoušejte knihovnu LibUSB pro obousměrnou komunikaci přes USB s vybraným mikropočítačem firmy Atmel. Komunikace musí umožňovat nejen režim dotaz-odpověď, ale i automatické vysílání dat z mikropočítače a naopak - zpracování událostí, přerušení a komunikaci s více mikroprocesory zároveň.

1. Analyzujte dostupné verze knihovny LibUSB pro platformu Win32 a prostudujte její možnosti.
2. Vyberte vhodný miktropočítač Atmel s rozhraním USB a seznamte se s programováním tohoto rozhraní. Najděte dostupné implementace programové USB vrstvy a vyberte jednu nejvhodnější.
3. Navrhněte programové rozhraní pro řídící počítač i mikropočítač, aby byla možná synchronní i asynchronní komunikace. Uvažujte i připojení více USB zařízení současně.
4. Implementujte navržené programové řešení a ověřte na prototypu jednoduché aplikace všechny režimy práce.
5. Otestujte stabilitu, propustnost a spolehlivost navrženého řešení.
6. Zhodnocení a přínosy práce.

Seznam doporučené odborné literatury:

[1] http://www.libusb.org
[2] http://www.atmel.com

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení:

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

7. 5. 2013

Lukáš Patočk

Abstrakt:

Předmětem práce je navrhnout a vytvořit programové vybavení pro mikrokontrolér firmy Atmel a aplikaci počítače v prostředí Win32 s využitím knihovny LibUSB, které společně umožní vzájemnou komunikaci přes USB rozhraní.

Práce obsahuje teoretickou část, která je zaměřena na výběr verze knihovny LibUSB, volbu nejvhodnějšího mikrokontroléru a programovou podporu USB vrstvy na straně mikrokontroléru. V teoretické části je také zahrnut stručný popis USB standardu. Praktická část obsahuje popis implementace softwaru, který umožní komunikaci počítače s mikrokontrolérem v synchronním i asynchronním režimu a umožní aplikaci počítače rozpoznat více připojených zařízení. Výsledek práce bude ověřen měřením přenosové rychlosti, spolehlivosti a chybovosti USB přenosů.

Klíčová slova:

Atmel AVR, AT90USB1287 ,deskriptor, enumerace, koncový bod, LibUSB, LibUSBx, LUFA, mikrokontrolér, USB- universální sériová sběrnice, synchronní komunikace, asynchronní komunikace

Abstract:

Subject of this work is to develop and create software for Atmel microcontroller together with Win32 application using LibUSB library, which will allow mutual communication via USB.

Work contents theoretical part, which is focused on selection of LibUSB library, choice of most suitable microcontroller and software support of USB layer on side of microcontroller. In theoretical part there is also included brief description of USB standard. Practical part contents description of implementation of software, which enables synchronous and asynchronous communication between microcontroller and computer and allows the computer application to recognize more connected devices. Result of this work will be verified by measurement of transfer speed, reliability and error rate of USB transfers.

Key words:

Atmel AVR, AT90USB1287, descriptor, enumeration, endpoint, LibUSB, LibUSBx, LUFA, microcontroller, USB- universal serial bus, synchronous communication, asynchronous communication

Seznam použitých symbolů a zkratek:

API	-	Application Programming Interface, programátorské rozhraní.
EEPROM	-	Electrically Erasable Programmable Read-Only Memory, elektricky mazatelná paměť typu ROM.
Flash (paměť)	-	Elektricky programovatelná paměť s náhodným přístupem.
Framework	-	Softwarová podpora při programování a vývoji softwarových projektů.
MFC	-	Microsoft Foundation Classes, knihovna, která zabaluje části Windows API do C++ tříd
Open-source	-	Software s legálně dostupným zdrojovým kódem.
SRAM	-	Static Random Access Memory, statická RAM paměť

1 Úvod 7

2 Teoretický rozbor 8

2.1 Knihovny LibUSB: 8

2.2 Mikrokontroléry Atmel AVR 9

2.3 Knihovny podporující USB komunikaci mikrokontrolérů Atmel 12

2.4 Specifikace USB standardu 12

3 Aplikace mikrokontroléru 16

3.1 Požadavky na funkci mikrokontroléru 16

3.2 Inicializace zařízení a knihovny LUFA 16

3.3 Vytvoření deskriptorů 16

3.4 Inicializace koncových bodů 20

3.5 Přenosy dat 21

4 Aplikace hostitelského PC 26

4.1 Inicializace USB zařízení 26

4.2 Kontrolní přenosy 29

4.3 Synchronní přenosy 29

4.4 Asynchronní přenosy 30

4.5 Popis hostitelské aplikace 33

4.6 Měření 34

5 Závěr 38

5.1 Zhodnocení naměřených výsledků 38

6 Zdroje: 39

7 Seznam tabulek 40

8 Seznam obrázků 41

9 Obsah CD 42

1 Úvod

Pod označením mikrokontrolér se zpravidla skrývá monolitický integrovaný obvod obsahující kompletní mikropočítač. Mikrokontroléry se vyznačují velkou spolehlivostí a kompaktností, většinou bývají využity v průmyslu, v oblastech řízení a regulace, své využití však najdou i v oblasti řízení například domácích spotřebičů.

Do dnes je pro komunikaci s mikrokontrolérem hojně využívána již poměrně zastaralá komunikace prostřednictvím sériové linky. Většina dnes prodávaných PC nebo notebooků již nejsou tímto rozhraním vybaveny. Využití USB může přinést spoustu nových využití mikrokontrolérů například v multimédiích nebo v oblasti domácí automatizace. Některé mikrokontroléry jsou schopny pracovat i v režimu USB hostitele, takže můžou řídit složitější USB periferní zařízení.

USB komunikace je schopná přenášet jednoduché řídící zprávy s garantovanou dobou doručení, což je využitelné pro aplikaci s interaktivním řízením, kde je zařízení ovládáno výstupy mikrokontroléru a mikrokontrolér může být ovládán uživatelem přes aplikaci v počítači. Kromě těchto zpráv mohou být přenášeny i velké bloky dat, což by bylo využitelné zejména v multimédiích, k přenosu obrazu nebo zvuku. Tyto dva typy zpráv se vzájemně neovlivňují, proto mohou být přenášeny současně.

Cílem této práce je prověřit možnosti mikrokontrolérů v oblasti komunikace přes USB rozhraní. Je tedy nutné zvolit vhodný mikrokontrolér z řady výrobků Atmel AVR, prověřit možnosti realizace programové USB vrstvy pro tento mikrokontrolér a vytvořit funkční program mikrokontroléru, na kterém bude možné demonstrovat schopnosti USB komunikace. Součástí práce je i aplikace pro hostitelský počítač, která pro podporu USB komunikace využívá knihovnu LibUSB a funguje v prostředí Win32. Úkolem je prověřit současný stav knihovny a s jejím využitím vytvořit hostitelskou aplikaci, která bude schopna pracovat v synchronním i asynchronním režimu a bude schopna ovládat více zařízení současně.

Výsledkem práce tedy bude funkční program pro obsluhu mikrokontroléru a hostitelská aplikace pro PC. Průběh komunikace mezi PC a mikrokontrolérem bude ověřen měřením chybovosti, spolehlivosti a rychlosti.

Práce je rozdělena do tří základních oddílů. První část obsahuje teoretický rozbor- zkoumá možnosti realizace USB vrstev pro PC i mikrokontrolér, volbu nejvhodnějšího mikrokontroléru a základní informace o USB protokolu. Druhá část popisuje vývoj programu mikrokontroléru. Třetí část obsahuje popis vývoje aplikace pro PC, její popis a výsledky měření.

2 Teoretický rozbor

V této části bakalářské práce budou rozebrány možnosti realizace USB rozhraní na straně PC, výběr nejvhodnějšího mikrokontroléru a programového vybavení pro implementaci USB vrstvy na straně kontroléru. Dále popis využitých funkcí USB protokolu.

2.1 Knihovny LibUSB:

LibUSB[1] je opensource knihovna jazyka C/C++, která umožňuje aplikacím přístup k USB zařízením na mnoha různých operačních systémech. Knihovna je šiřitelná pod licencí GNU. LibUSB vyšla zatím ve dvou verzích s vzájemně nekompatibilními API:

- LibUSB 0.1
- LibUSB 1.0

2.1.1 LibUSB 0.1 - LibUSB-win32

LibUSB 0.1[2] je původní, dnes již nepodporovaná knihovna vycházející od roku 2000 do roku 2006 vytvořená za účelem umožnění přístupu k USB zařízení. Tato knihovna je podporována operačními systémy Linux, FreeBSD, NetBSD, OpenBSD, Darwin/MacOS X a Solaris.

Pro operační systém Microsoft Windows byla vytvořena odnož knihovny LibUSB 0.1 nazvaná LibUSB-win32[3]. Knihovna LibUSB-win32 je podporována systémy Windows počínaje Windows 98 SE až po Windows 7, podporuje všechny typy USB přenosů (Control, Bulk, Interrupt, Isochronous) a všechny typy standardních i uživatelsky definovaných kontrolních zpráv.

2.1.2 LibUSB 1.0 - Windows backend

LibUSB 1.0 vychází od roku 2008, jedná se o rozšířený přepis stabilní verze LibUSB 0.1. Oproti knihovně LibUSB 0.1 je doplněna o možnost asynchronní komunikace a isochronní přenosy, podporu více vláken a jednoduché API. LibUSB 1.0 je zpětně kompatibilní s LibUSB 0.1 s využitím mezivrstvy LibUSB-compat-0.1. Knihovna je v současnosti označena jako kompletní a veškeré úpravy se týkají pouze opravy chyb. S dalšími vylepšeními a novými funkcemi se počítá až v nadcházející verzi 1.1.

Jako podpora platformy Windows vzniká projekt LibUSB 1.0 Windows Backend[4]. Od verze 1.0.9 je knihovna označována jako stabilní a podporuje všechny systémy Windows počínaje Windows XP, kromě Windows 2003 a Windows XP 64 bit, v těchto systémech není podporován ovladač WinUSB. Od této verze jsou také podporovány zařízení USB 3.0, i když v experimentálním stádiu. Knihovna LibUSB 1.0 Windows Backend má několik nedostatků:

- Není možné nastavit jinou než defaultní konfiguraci zařízení.
- Knihovna nepodporuje více aplikací najednou.
- Nejsou podporovány izochronní přenosy.

2.1.3 LibUSBx

Knihovna LibUSBx[5] vzniká odtržením vývojové skupiny od tvůrců knihovny LibUSB z důvodu jejich neaktivity. LibUSBx je přenositelná mezi systémy Linux, OS X, Windows a OpenBSD a podporuje všechny USB standardy od 1.0 do 3.0. Rozhraní knihovny LibUSBx je kompatibilní s knihovnou LibUSB. Autoři této knihovny slibují aktivní přístup k vývoji knihovny a časté vydání nových verzí (viz [6]). V současné době platí pro knihovnu LibUSBx stejné nedostatky jako pro knihovnu LibUSB 1.0.

Pro fungování knihovny je nutná instalace některého z ovladačů: WinUSB, libusbK nebo libusb0.

Vývojová prostředí podporovaná knihovnou LibUSBx:

- MinGW (32 bit) a MinGW-w64
- Microsoft Visual C++ (Visual Studio)
- Windows DDK

2.2 Mikrokontroléry Atmel AVR

Atmel AVR[7] je označení pro rodinu 8 a 32 bitových mikrokontrolerů typu RISC, které poskytují jedinečnou kombinaci výkonu a nízkých nároků na napájení, jejich architektura je optimalizována pro efektivitu kódu psaného v jazyce C. AVR mikrokontroléry jsou navrženy podle upravené Harvardské architektury- program a data jsou uložena v různých paměťových prostorech. Pro uložení programu je využita flash paměť, data jsou uložena v EEPROM, navíc je možné pomocí speciální instrukce přistupovat k datům uloženým ve flash paměti. Pro běh aplikací je k dispozici SRAM. Takt mikrokontrolérů AVR je pohybuje až do 32MHz.

Přehled typů mikrokontrolérů AVR:

- **Tiny AVR**
  - 0,5 - 16 kB programové paměti
  - 6 až 32 pinů pouzdra
- **Mega AVR**
  - 4 - 512 kB programové paměti
  - 28 - 100 pinů pouzdra
  - rozšířená sada instrukcí
  - velký počet periferních zařízení

- **XMEGA**
  - 16-384 kB programové paměti
  - 44 - 100 pinů pouzdra
  - rozšířeny o některé funkce (například DMA, podpora kryptografie)
  - velký počet periferních zařízení

Některé z mikrokontrolérů Atmel AVR obsahují USB port. Jsou to například tyto typy:

- **32-bit AVR UC3:**
  ATUC64L3U, ATUC128L3U, ATUC256L3U,
  ATUC64L4U, ATUC128L4U, ATUC256L4U
- **mega AVR:**
  ATmega8U2, ATmega16U2, ATmega32U2, ATmega16U4, ATmega32U4,
  řada AT90USB
- **AVR XMEGA:**
  ATxmega64A1U, ATxmega128A1U,
  ATxmega64A3U, ATxmega128A3U, ATxmega192A3U, ATxmega256A3U,
  ATxmega256A3BU,
  ATxmega16A4U, ATxmega32A4U, ATxmega64A4U, ATxmega128A4U

2.2.1 Řada mikrokontrolérů AT90USB

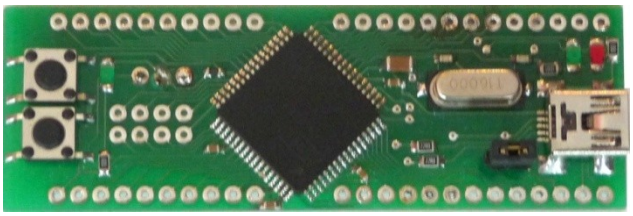
Typ	paměť flash	paměť EEPROM	paměť SRAM	počet pinů	USB hostitel	počet konc. bodů
AT90USB82	8kB	512B	512B	32	NE	4+1
AT90USB162	16kB	512B	512B	32	NE	4+1
AT90USB646	64kB	2kB	4kB	64	NE	6+1
AT90USB647	64kB	2kB	4kB	64	ANO	6+1
AT90USB1286	128kB	4kB	8kB	64	NE	6+1
AT90USB1287	128kB	4kB	8kB	64	ANO	6+1

Tabulka 1 - Přehled mikrokontrolérů řady AT90USB

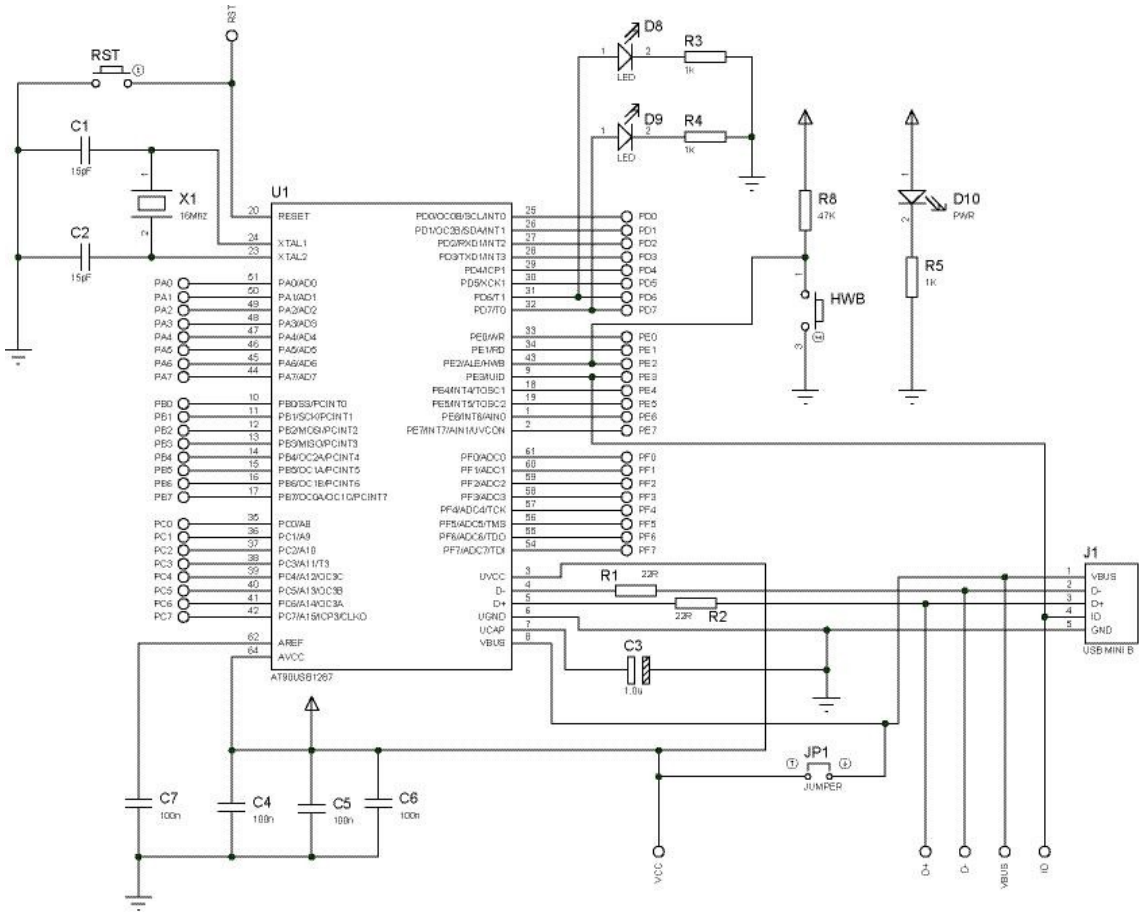
Pro realizaci práce byl zvolen mikrokontrolér AT90USB1287[8]. Je to nejlépe vybavený zástupce řady AT90USB, nabízí podporu USB jako zařízení i jako hostitele. Tento mikrokontrolér pracuje na frekvenci 16MHz, Full Speed USB pracuje na 48MHz, tato frekvence je získána pomocí PLL násobičky. Mikrokontrolér nabízí kontrolní koncový bod o velikosti 64B a dalších 6 programovatelných koncových bodů. Pro těchto 6 koncových bodů je vyhrazena paměť SRAM o velikosti 832B.

Pro realizaci práce byla využita zapůjčená deska plošného spoje s tímto mikrokontrolérem, která je zobrazena na obrázku 1, její schéma je na obrázku 2. Deska obsahuje dvě tlačítka, první má funkci

reset, druhé slouží v okamžiku startu zařízení ke spuštění bootloaderu, za běhu aplikace lze jeho funkci programově definovat. Dále deska plošného spoje obsahuje tři LED diody, první dioda zelené barvy indikuje stav zapnutí, další dvě (zelená a červená) jsou ovladatelné programem. Ostatní vstupně/výstupní piny jsou vyvedeny na okraj desky plošného spoje pro případné využití externími zařízeními.



Obrázek 1 - Deska plošného spoje



Obrázek 2 - schéma zapojení

2.3 Knihovny podporující USB komunikaci mikrokontrolérů Atmel

- **AVR USB device stack**

Jedná se o podporu USB komunikace, která je součástí ASF (Atmel® AVR Software Framework), vyvíjené společností Atmel.

- **knihovna LUFA (Lightweight USB Framework for AVRs)**

LUFA[9] je kompletní open-source (s volně dostupným zdrojovým kódem) framework pro podporu 8bitových a některých 32bitových mikrokontrolérů AVR s USB rozhraním. Je šiřitelná pod licencí MID, tato licence zaručuje její volné využití i šíření. Framework LUFA obsahuje vlastní bootloader pro mikrokontroléry AVR a několik demo projektů (např. virtuální sériový port). Programy napsané s využitím knihovny LUFA by měly být přenositelné mezi všemi mikrokontroléry Atmel AVR. Podrobné srovnání knihovny LUFA s Atmel USB device stack je najdete na stránce [10]

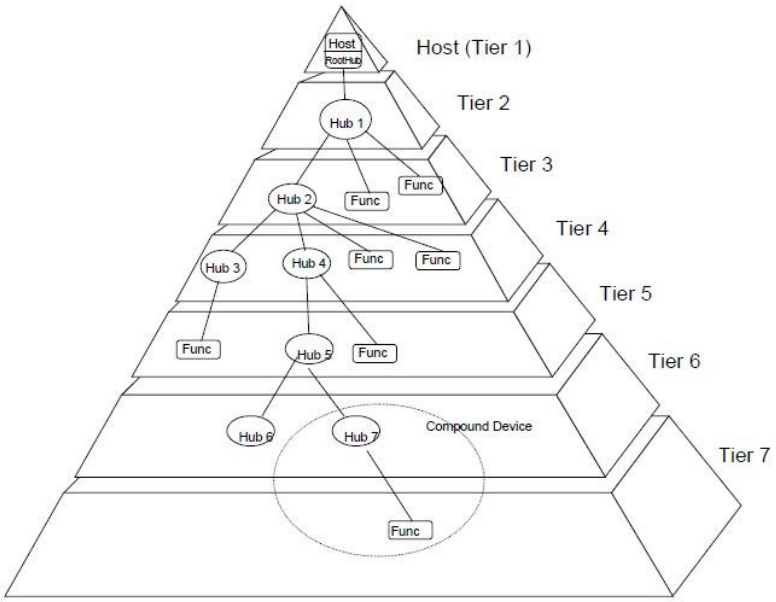
2.4 Specifikace USB standardu

USB (univerzální sériová sběrnice)[11] je specifický standard rozšiřující PC architekturu se zaměřením na periferní zařízení. Mezi nejvýznamnější vlastnosti tohoto standardu patří:

- Jednoduchost připojení a použití periferních zařízení (plug and play).
- Cenově nenáročné řešení podporující vysokou přenosovou rychlost.
- Podpora přenosu obrazu a zvuku v reálném čase.
- Umožňuje kombinaci isochronního datového přenosu a zasílání asynchronních zpráv.
- Poskytuje standardní rozhraní umožňující snadnou integraci nových produktů.
- Připojení až 127 periferií současně.

Verze	USB standardy	Přenosová rychlost
USB 1.1	low speed full speed	1,5 Mb/s 12 Mb/s
USB 2.0	high speed	480 Mb/s
USB 3.0	super speed	4800 Mb/s

Tabulka 2 - Přehled USB standardů



Obrázek 3- Hierarchie USB zařízení

2.4.1 Typy USB přenosů

Kontrolní přenosy (control transfers)

Tyto přenosy využívá především systémový software pro konfiguraci zařízení při jeho prvotním připojení. Kontrolní přenosy mohou být využity i uživatelským softwarem specifickým způsobem. Spolehlivost předávání dat kontrolními přenosy je zajištěna potvrzováním.

Přenosy přerušení (interrupt transfers)

Přenosy s garantovaným časem doručení, využívají se v aplikacích, které zpracovávají události vzniklé buď na straně hostitele, nebo zařízení. Přenášená zpráva většinou obsahuje informaci o typu vzniklé události a malý blok dat. Využívají se například u polohovacích zařízení k přenosu souřadnic.

Hromadné přenosy (bulk transfers)

Přenášená data se obvykle skládají z větších bloků dat, využívají se například pro tiskárny a skenery. Data jsou přenášena sekvenčně, spolehlivost přenosu dat je zajištěna na hardwarové úrovni pomocí detekce chyb a vyvolání omezeného počtu opakovaných odeslání. Šířka pásma pro hromadné přenosy může být ovlivněna ostatními přenosy na sběrnici.

Izochronní přenosy (isochronous transfer)

Izochronní data jsou přenášena nepřetržitě, spojitě po celou dobu trvání přenosu. Jsou to data, u kterých je požadováno minimální zpoždění jejich doručení, jako je například telefonní hovor, přenos obrazu nebo zvuku. Izochronní přenosy probíhají na vyhrazené části přenosového pásma a díky tomu jsou

data přenášena konstantní rychlostí. Izochronní přenosy nezaručují bezchybnost přenášených dat, předpokládá se, že vzniklé chyby budou v tak zanedbatelném množství, že nebudou mít výrazný vliv na kvalitu přenosu.

2.4.2 Koncové body (endpoint)

Koncový bod je jednoznačný identifikátor části USB zařízení, který specifikuje adresu komunikačního kanálu mezi hostitelem a zařízením. Každý koncový bod má charakteristické vlastnosti:

- Číslo koncového bodu.
- Směr přenosu.
- Typ přenosu.
- Maximální velikost přenášeného bloku dat.
- Požadavek na šířku přenosového pásma.
- Požadavek na frekvenci dotazování nebo zpoždění přenosu.

USB zařízení může obsahovat až 32 koncových bodů a v každém zařízení musí být implementovány dva výchozí koncové body pro obsluhu kontrolních přenosů. Tyto dva koncové body jsou pro zjednodušení oba identifikovány číslem 0, pro uživatele se tedy tváří jako jeden koncový bod komunikující v obou směrech. Zbylých 30 koncových bodů může být využito uživatelem, 15 ve směru od hostitele k zařízení a 15 ve směru od zařízení k hostiteli. Směr od zařízení k hostiteli je vždy označován jako IN a od hostitele k zařízení OUT.

2.4.3 Roury (pipe)

Roura je označení logického spojení mezi koncovým bodem na straně zařízení a softwarem na straně hostitele. Roura reprezentuje schopnost přesunu dat mezi softwarem na straně hostitelem a vyrovnávací pamětí zařízení. Existují dva typy rour:

- Datový tok (stream) – data jsou přenášena nepřetržitě v nespecifikovaném proudu (izochronní přenosy).
- Zpráva (message) – data jsou přenášena v přesně definované struktuře v pevně daném intervalu, nebo v reakci na vzniklou událost (bulk a interrupt přenosy).

2.4.4 Deskriptory (descriptor)

USB zařízení identifikuje své charakteristické vlastnosti pomocí definované struktury zvané deskriptor. Každý deskriptor obsahuje hlavičku nesoucí informaci o celkové velikosti deskriptoru a jeho typu. Další data jsou závislá na typu deskriptoru:

- Deskriptor zařízení (Device descriptor)

  Obsahuje informace o zařízení:
  - verze USB specifikace
  - třída zařízení
  - protokol
  - velikost výchozích koncových bodů
  - Vendor ID a Product ID
  - sériové číslo
  - atd.…
- Deskriptor konfigurace (Configuration descriptor)

  Obsahuje informace o konfiguraci, deskriptor rozhraní a deskriptory jednotlivých koncových bodů:
  - číslo konfigurace
  - počet rozhraní
  - maximální proud napájení
- Deskriptor rozhraní (Interface descriptor)

  Obsahuje informace o rozhraní:
  - číslo rozhraní.
  - počet koncových bodů.
- Deskriptor koncového bodu (Endpoint descriptor)

  Obsahuje informace o koncovém bodu:
  - Adresa koncového bodu.
  - Vlastnosti (např. typ koncového bodu).
  - Velikost koncového bodu.
- String descriptor

  Obsahuje textový popis zařízení.

2.4.5 Enumerace

Enumerace je proces, který nastává v okamžiku připojení USB zařízení k hostiteli. USB hub informuje hostitele o připojení nového zařízení, hostitel načte informaci o portu, na který bylo zařízení připojeno, a povolí tento port. Hostitel načte informace z deskriptoru zařízení, přidělí zařízení adresu a načte informace o konfiguraci zařízení. Podle načtených informací z deskriptoru také hostitel zvolí ovladač, který bude pro zařízení využit. Po dokončení enumerace je zařízení připraveno k použití.

3 Aplikace mikrokontroléru

Následující kapitola bude obsahovat popis implementace software pro mikrokontrolér. Pro implementaci USB vrstvy aplikace mikrokontroléru byla vzhledem k množství dostupné dokumentace a příkladů použití zvolena knihovna LUFA. V současnosti je vydán stabilní release LUFA 130303.

3.1 Požadavky na funkci mikrokontroléru

- Hlavním úkolem práce je otestovat USB přenosy.
- Software mikrokontroléru musí být schopen provést enumeraci- odeslat hostitelskému zařízení informace v podobě deskriptorů.
- Pro identifikaci více připojených zařízení je možnost na vyžádání nastavit či odeslat identifikátor zařízení.
- Zařízení musí reagovat na stisk tlačítka a odeslat o této události informaci hostitelské aplikaci.
- Naopak kontrolér musí reagovat zablikáním LED diody na událost generovanou hostitelem.
- Kontrolér musí být schopen zpracovávat příchozí i odchozí bulk, interrupt a izochronní přenosy.

3.2 Inicializace zařízení a knihovny LUFA

Použitá deska plošného spoje obsahuje dvě programem řiditelné LED diody a dvě tlačítka. První tlačítko má funkci reset a druhé HWB, toto tlačítko stisknuté při startu zařízení nebo resetu spouští bootloader, ale za běhu aplikace je možné jeho funkci programově definovat. Inicializace a funkce pro využití těchto periferií se nacházejí v souborech `io.h` a `io.c`. Jsou zde definovány funkce pro ovládání LED diod a pro zjištění stavu tlačítka. Při inicializaci je nutné programově nastavit připojení pull-up rezistoru k pinu, na který je připojeno tlačítko, aby nedocházelo k zákmitům a rušení.

Aby bylo možné využívat funkce knihovny LUFA pro podporu USB, je nutné do zdrojových kódů připojit direktivu `#include <LUFA/Drivers/USB/USB.h>` a k projektu připojit knihovny `lufa_core.mk`, `lufa_sources.mk`, `lufa_build.mk`. Samotná inicializace knihovny LUFA při startu zařízení je provedena voláním funkce `USB_Init()`, před kterým je ještě nutné povolit přerušení zavoláním funkce `sei()`.

3.3 Vytvoření deskriptorů

Pro potřebu enumerace připojeného zařízení je nutné do zařízení uložit informace o zařízení a možnostech jeho konfigurace. K uložení těchto informací slouží předdefinované datové struktury.

- Deskriptor zařízení - USB_Descriptor_Device_t
- Deskriptor konfigurace je struktura složená z:

- hlavičky konfigurace - USB_Descriptor_Configuration_Header_t
- deskriptoru rozhraní - USB_Descriptor_Interface_t
- deskriptorů koncových bodů - USB_Descriptor_Endpoint_t

3.3.1 Deskriptor zařízení

Deskriptor zařízení identifikuje zařízení ihned po jeho připojení. Předává hostiteli informaci o tom, kterou verzi USB standardu zařízení používá. Na základě informací uložených v tomto deskriptoru je systémem zvolen ovladač, který bude pro toto zařízení použit. Každý deskriptor také obsahuje hlavičku, ve které je uložena velikost deskriptoru a jeho typ. Hlavička je typu `USB_Descriptor_Header_t`. Ukázka vytvoření deskriptoru zařízení:

```
const USB_Descriptor_Device_t PROGMEM DeviceDescriptor =
{
	.Header 		=	// hlavicka deskriptoru
	{
		.Size = sizeof(USB_Descriptor_Device_t), // velikost deskriptoru
		.Type = DTYPE_Device 			// typ deskriptoru
	},

	.USBSpecification   = VERSION_BCD(02.00),	// verze USB standardu
	.Class 		    = 0x00,			// trida USB zarizeni
	.SubClass 	    = 0x00,			// podtrida zarizeni
	.Protocol 	    = 0x00,			// pouzity protokol

			    // velikost kontrolniho koncoveho bodu
	.Endpoint0Size      = FIXED_CONTROL_ENDPOINT_SIZE,

	.VendorID 	    = 0x03EB,			// Vendor ID
	.ProductID 	    = 0x2065,			// Product ID
	.ReleaseNumber      = VERSION_BCD(01.00),	// verze

	.ManufacturerStrIndex	= 0x01,		// index string deskriptoru vyrobce
	.ProductStrIndex 	= 0x02,		// index string deskriptoru zarizeni
	.SerialNumStrIndex  = USE_INTERNAL_SERIAL,
			    // index string deskriptoru s vyrobnim cislem

	.NumberOfConfigurations = FIXED_NUM_CONFIGURATIONS    // pocet moznych konfiguraci
};
```

3.3.2 Deskriptor konfigurace

Deskriptor konfigurace nese informace o této konfiguraci, jejich rozhraních a koncových bodech. Pro uložení všech informací obsažených v deskriptoru konfigurace byla vytvořena datová struktura `USB_Descriptor_Configuration_t`, která obsahuje všechny zmiňované deskriptory. Definice struktury:

```
typedef struct
{
	USB_Descriptor_Configuration_Header_t Configuration;	// konfigurace

	USB_Descriptor_Interface_t Interface;			// rozhrani
	USB_Descriptor_Endpoint_t	InterruptInEndpoint;	// interrupt in endpoint
	USB_Descriptor_Endpoint_t	InterruptOutEndpoint;	// interrupt out endpoint
	USB_Descriptor_Endpoint_t	BulkInEndpoint;		// bulk in endpoint
	USB_Descriptor_Endpoint_t	BulkOutEndpoint;	// bulk out endpoint
	USB_Descriptor_Endpoint_t	IsochronousInEndpoint;	// isochronous in endpoint
	USB_Descriptor_Endpoint_t	IsochronousOutEndpoint;	// isochronous out endpoint
}	USB_Descriptor_Configuration_t;
```

USB_Descriptor_Configuration_Header_t		
Název	Hodnota	Popis
`TotalConfigurationSize`	`sizeof(USB_Descriptor_Configuration_t)`	Celková velikost konfiguračního deskriptoru
`TotalInterfaces`	`1`	Počet rozhraní
`ConfigurationNumber`	`0`	Číslo konfigurace
`ConfigurationStrIndex`	`NO_DESCRIPTOR`	Adresa řetězce s popisem konfigurace
`ConfigAttributes`	`(USB_CONFIG_ATTR_RESERVED | USB_CONFIG_ATTR_SELFPOWERED)`	Atributy konfigurace
`MaxPowerConsumption`	`USB_CONFIG_POWER_MA(100)`	Proudové zatížení

Tabulka 3 - Popis deskriptoru konfigurace

USB_Descriptor_Interface_t		
Název	Hodnota	Popis
`InterfaceNumber`	`0`	Číslo rozhraní
`AlternateSetting`	`0`	Alternativní nastavení
`TotalEndpoints`	`6`	Počet koncových bodů
`Class`	`0x00`	Třída zařízení
`SubClass`	`0x00`	Podtřída zařízení
`Protocol`	`0x00`	Použitý protokol
`InterfaceStrIndex`	`NO_DESCRIPTOR`	Adresa řetězce s popisem rozhraní

Tabulka 4 - Popis deskriptoru rozhraní

USB_Descriptor_Endpoint_t		
Název	Hodnota	Popis
Endpoint 1:		
EndpointAdress	EP_ISO_OUT	Adresa koncového bodu
Attributes	(EP_TYPE_ISOCHRONOUS | ENDPOINT_ATTR_NO_SYNC | ENDPOINT_USAGE_DATA)	Atributy koncového bodu: typ, synchronizace, využití
EndpointSize	EP_ISO_SIZE	Velikost vyrovnávací paměti
PollingIntervalMS	100	Časování
Endpoint 2:		
EndpointAdress	EP_ISO_IN	Adresa koncového bodu
Attributes	(EP_TYPE_ISOCHRONOUS | ENDPOINT_ATTR_NO_SYNC | ENDPOINT_USAGE_DATA)	Atributy koncového bodu: typ, synchronizace, využití
EndpointSize	EP_ISO_SIZE	Velikost vyrovnávací paměti
PollingIntervalMS	100	Časování
Endpoint 3:		
EndpointAdress	EP_BULK_OUT	Adresa koncového bodu
Attributes	(EP_TYPE_ISOCHRONOUS | ENDPOINT_ATTR_SYNC | ENDPOINT_USAGE_DATA)	Atributy koncového bodu: typ, synchronizace, využití
EndpointSize	EP_BULK_SIZE	Velikost vyrovnávací paměti
PollingIntervalMS	100	Časování
Endpoint 4:		
EndpointAdress	EP_BULK_IN	Adresa koncového bodu
Attributes	(EP_TYPE_ISOCHRONOUS | ENDPOINT_ATTR_SYNC | ENDPOINT_USAGE_DATA)	Atributy koncového bodu: typ, synchronizace, využití
EndpointSize	EP_BULK_SIZE	Velikost vyrovnávací paměti
PollingIntervalMS	100	Časování
Endpoint 5:		
EndpointAdress	EP_INT_OUT	Adresa koncového bodu
Attributes	(EP_TYPE_ISOCHRONOUS | ENDPOINT_ATTR_SYNC | ENDPOINT_USAGE_DATA)	Atributy koncového bodu: typ, synchronizace, využití
EndpointSize	EP_INT_SIZE	Velikost vyrovnávací paměti
PollingIntervalMS	100	Časování
Endpoint 6:		
EndpointAdress	EP_INT_IN	Adresa koncového bodu
Attributes	(EP_TYPE_ISOCHRONOUS | ENDPOINT_ATTR_SYNC | ENDPOINT_USAGE_DATA)	Atributy koncového bodu: typ, synchronizace, využití
EndpointSize	EP_INT_SIZE	Velikost vyrovnávací paměti
PollingIntervalMS	100	Časování

Tabulka 5 - Popis deskriptorů koncových bodů

Konstanta	Hodnota
EP_ISO_OUT	0x01
EP_ISO_IN	0x82
EP_BULK_OUT	0x03
EP_BULK_IN	0x84
EP_INT_OUT	0x05
EP_INT_IN	0x86
EP_ISO_SIZE	8
EP_BULK_SIZE	64
EP_INT_SIZE	256

Tabulka 6 - Hodnoty konstant koncových bodů

3.3.3 Odeslání deskriptorů hostiteli

Po připojení zařízení si systém hostitele vyžádá odeslání jednotlivých deskriptorů (proces enumerace). Obslužný software musí být schopen na tyto požadavky reagovat a hostiteli požadované deskriptory odeslat. Požadavek na zaslání deskriptoru je zpracován nízkoúrovňovou rutinou knihovny LUFA, která následně volá funkci `CALLBACK_USB_GetDescriptor`.

```
uint16_t CALLBACK_USB_GetDescriptor(
          const uint16_t wValue,
          const uint8_t wIndex,
          const void **const DescriptorAddress)
```

Funkci jsou předány parametry rozhodující o tom, který deskriptor se má hostiteli odesílat, na jejich základě vybere požadovaný deskriptor a vrací informace o něm - výstupním parametrem předává ukazatel na tento deskriptor a návratovou hodnotou jeho velikost.

3.4 Inicializace koncových bodů

Před zahájením vlastní komunikace na jiném než kontrolním koncovém bodě musí hostitel zvolit konfiguraci zařízení. Po zvolení dané konfigurace probíhá inicializace koncových bodů využitých v této konfiguraci. Požadavek na nastavení konfigurace opět zpracovává nízkoúrovňová rutina knihovny LUFA, která následně vyvolá událost reprezentovanou voláním funkce `void EVENT_USB_Device_ConfigurationChanged()`, ve které je provedena inicializace jednotlivých koncových bodů voláním funkce `Endpoint_ConfigureEndpoint`.

```
static bool Endpoint_ConfigureEndpoint(
          const uint8_t Address,
          const uint8_t Type,
          const uint16_t Size,
          const uint8_t Banks )
```

3.5 Přenosy dat

Protokol USB poskytuje několik druhů datových přenosů. Základním typem je přenos standardních zpráv na kontrolní koncový bod, tyto zprávy jsou zpracovávány nízkoúrovňovými rutinami knihovny LUFA a programátor k nim nemá přístup, využívají se při enumeraci a změnách konfigurace zařízení. Kontrolní koncový bod je schopen zpracovávat i uživatelsky definované nestandardní zprávy. Dále protokol USB nabízí přerušovací přenosy, bulk přenosy a isochronní přenosy.

3.5.1 Uživatelsky definované kontrolní přenosy

Při přijetí jiné než standardní zprávy na kontrolní koncový bod je vyvolána událost reprezentovaná funkcí `EVENT_USB_Device_ControlRequest`. A veškeré informace o této zprávě jsou uloženy v globální proměnné `USB_ControlRequest`, která je typu `USB_Request_Header_t`.

USB_Request_Header_t	
Název	Popis
bmRequestType	typ požadavku- směr, typ a příjemce
bRequest	kód požadavku
wIndex	volitelný parametr přenosu
wLength	délka připojených dat
wValue	volitelný parametr přenosu

Tabulka 7 - Popis uživatelsky definovaných kontrolních zpráv

Funkce `EVENT_USB_Device_ControlRequest` má za úkol identifikovat přijatou zprávu a následně na ní zareagovat. Po přijetí zprávy na kontrolní koncový bod je nutné potvrdit toto přijetí zavoláním funkce `Endpoint_ClearSETUP()`, která odesílá hostiteli informaci o tom, že byla kontrolní zpráva úspěšně přijata. Zpráva je identifikována na základě hodnoty bRequest a hodnota bmRequestType může nést informace o směru přenosu dat a jejím typu. Uživatelsky definované zprávy mohou obsahovat data. V takovém případě je nutné k jejich načtení nebo odeslání využít některou z funkcí knihovny LUFA viz tabulka 9. K dispozici jsou funkce pro načtení nebo odeslání zprávy definované délky, tyto funkce jsou rozděleny podle typu paměti, se kterou pracují, a podle formátu dat (little endian / big endian). Po odeslání či přijetí dat je třeba opět potvrdit tuto událost voláním funkce Endpoint_ClearIN pro odeslaná data a Endpoint_ClearOUT pro přijatá data.

Funkce pro zápis a čtení dat z paměti RAM jsou označeny slovy "Control_Stream", pro práci s pamětí EEPROM jsou funkce označeny "Control_EStream" a pro čtení z FLASH paměti "Control_PStream", do této paměti není umožněno zapisovat. Dále jsou tyto funkce rozděleny příponami "_BE" pro big endian a "_LE" pro little endian. Směr komunikace je určen slovem "Read" pro načtení dat přijatých od hostitele a "Write" pro zapsání dat pro odeslání hostiteli.

Prototypy funkcí pracujících s kontrolním koncovým bodem:

```
uint8_t Endpoint_Write_Control_Stream_LE(const void *const Buffer, uint16_t Length);
uint8_t Endpoint_Write_Control_Stream_BE(const void *const Buffer, uint16_t Length);
uint8_t Endpoint_Read_Control_Stream_LE(void *const Buffer, uint16_t Length);
uint8_t Endpoint_Read_Control_Stream_BE(void *const Buffer, uint16_t Length);
uint8_t Endpoint_Write_Control_EStream_LE(const void *const Buffer, uint16_t Length);
uint8_t Endpoint_Write_Control_EStream_BE(const void *const Buffer, uint16_t Length);
uint8_t Endpoint_Read_Control_EStream_LE(void *const Buffer, uint16_t Length);
uint8_t Endpoint_Read_Control_EStream_BE(void *const Buffer, uint16_t Length);
uint8_t Endpoint_Write_Control_PStream_LE(const void *const Buffer, uint16_t Length);
uint8_t Endpoint_Write_Control_PStream_BE(const void *const Buffer, uint16_t Length);
```

Parametry:

Buffer - Blok dat se kterým funkce pracují.

Length - Velikost přijatých dat, nebo dat k odeslání.

3.5.2 Použité kontrolní zprávy

Název	Kód	Popis
REQ_GET_ID	0xA0	Odeslání ID zařízení
REQ_SET_ID	0xA1	Nastavení ID zařízení
REQ_SET_SEND	0xA2	Nastavení zařízení do režimu odesílání dat
REQ_SET_RECEIV	0xA3	Nastavení zařízení do režimu příjmu dat
REQ_SET_REPEAT	0xA4	Nastavení zařízení do režimu *echo*
REQ_GET_ERROR	0xA5	Požadavek na odeslání počtu chyb přenosu
REQ_SEND_INTS	0xA6	Požadavek na odeslání několika interrupt zpráv

Tabulka 8 - Seznam použitých kontrolních zpráv

3.5.3 Příjem dat pomocí bulk, interrupt a isochronních přenosů

V hlavní smyčce programu je opakovaně volána funkce `checkUSB`, která v každém cyklu programu ověřuje příchod dat na jednotlivé vstupní koncové body. Funkce knihovny LUFA pracující s koncovými body (příjem a odesílání dat, kontrola stavu atd.) nemají ve svých parametrech zadáno, s kterým koncovým bodem budou pracovat. Je tedy nutné před použitím takové funkce nastavit, který koncový bod má být použit. K tomu slouží funkce `Endpoint_SelectEndpoint`, má jediný parametr a tím je právě číslo koncového bodu, s kterým se chystáme pracovat. Po zvolení potřebného koncového bodu následuje volání funkce `Endpoint_IsOUTReceived`. Tato funkce nám ověří, jestli koncový bod obsahuje nějaká příchozí data. V případě, že byla na tento koncový bod přijata data, vrací funkce booleovskou hodnotu true, jinak false. V každém cyklu programu jsou tedy postupně prověřeny všechny vstupní koncové body a v případě příchozích dat je volá funkce pro obsluhu daného přenosu.

3.5.4 Funkce knihovny LUFA pro příjem dat

Funkce pro příjem dat jsou rozděleny na dva základní typy. Jsou to funkce pro příjem primitivních dat, které přijímají buď jeden byte (8 bitů), jeden word (16 bitů) nebo jeden doubleword(32 bitů), pro příjem 16 a 32 bitů jsou funkce ještě rozděleny podle formátu na little endian a big endian. Prototypy funkcí pro příjem primitivních dat:

```
static uint8_t Endpoint_Read_8 (void);
static uint16_t Endpoint_Read_16_LE (void);
static uint16_t Endpoint_Read_16_BE (void);
static uint32_t Endpoint_Read_32_LE (void);
static uint32_t Endpoint_Read_32_BE (void);
```

Druhou možností příjmu dat jsou funkce, které najednou přijímají blok dat, jehož velikost je definována v parametru. Tyto funkce jsou rozděleny totožně jako funkce pro čtení dat z kontrolního koncového bodu. Je to tedy rozdělení podle typu paměti a podle formátu ukládání dat. Funkce mají i totožný název, jen neobsahují slovo "Control" a mají navíc parametr `BytesProcessed` , což je ukazatel na číslo, které obsahuje počet celkem přenesených bytů při přenosu. Prototypy funkcí hromadného příjmu dat:

```
uint8_t Endpoint_Read_Stream_LE (void *const Buffer, uint16_t Length, uint16_t *const
BytesProcessed);
uint8_t Endpoint_Read_Stream_BE (void *const Buffer, uint16_t Length, uint16_t *const
BytesProcessed);
uint8_t Endpoint_Read_EStream_LE (void *const Buffer, uint16_t Length, uint16_t *const
BytesProcessed);
uint8_t Endpoint_Read_EStream_BE (void *const Buffer, uint16_t Length, uint16_t *const
BytesProcessed);
```

3.5.5 Funkce knihovny LUFA pro odesílání dat

O funkcích pro odesílání dat platí stejná pravidla jako pro příjem dat, proto nyní uvádím pouze jejich prototypy:

```
static void Endpoint_Write_8 (const uint8_t Data);
static void Endpoint_Write_16_LE (const uint16_t Data);
static void Endpoint_Write_16_BE (const uint16_t Data);
static void Endpoint_Write_32_LE (const uint32_t Data);
static void Endpoint_Write_32_BE (const uint32_t Data);
```

```
uint8_t Endpoint_Write_Stream_LE (const void *const Buffer, uint16_t Length, uint16_t
*const BytesProcessed);
uint8_t Endpoint_Write_Stream_BE (const void *const Buffer, uint16_t Length, uint16_t
*const BytesProcessed);
uint8_t Endpoint_Write_EStream_LE (const void *const Buffer, uint16_t Length, uint16_t
*const BytesProcessed);
uint8_t Endpoint_Write_EStream_BE (const void *const Buffer, uint16_t Length, uint16_t
*const BytesProcessed);
uint8_t Endpoint_Write_PStream_LE (const void *const Buffer, uint16_t Length, uint16_t
*const BytesProcessed);
uint8_t Endpoint_Write_PStream_BE (const void *const Buffer, uint16_t Length, uint16_t
*const BytesProcessed);
```

3.5.6 Interrupt přenosy

V testovacím programu byly tyto přenosy využity k přenosu zpráv, které v reálném čase ovlivňují chování mikrokontroléru na základě události vzniklé v hostitelské aplikaci a opačně. Hostitelská aplikace obsahuje položku *„Počet stisknutí tlačítka“*, která v průběhu běhu aplikace počítá, kolikrát bylo tlačítko stisknuto. Na druhou stranu hostitelská aplikace umožňuje tlačítky rozsvítit a zhasnout LED diodu na zařízení. Pro otestování spolehlivosti přerušovacích přenosů bylo ještě do aplikací přidáno obousměrné hromadné odeslání a přijetí přerušovacích zpráv.

V okamžiku přijetí zprávy na přerušovacím koncovém bodě tuto událost ohlásí funkce `checkUSB` a volá následně funkci `interrupt_IO`. Ta pomocí volání funkce knihovny LUFA `Endpoint_BytesInEndpoint` zjistí, jak velký blok dat byl koncovým bodem přijat. Zjištěná velikost dat je následně použita jako parametr funkce `Endpoint_Read_Stream_LE`, která načte přijatá data. Na straně mikrokontroléru jsou zpracovávány 3 typy zprávy. Jedna rozsvítí kontrolní LED diodu, druhá ji zhasne a třetí zvyšuje počítadlo zpráv, které je pak možné vyčíst kontrolním přenosem pro vyhodnocení spolehlivosti interrupt přenosů při hromadném přijetí na straně kontroléru.

K odeslání interrup zprávy dochází v okamžiku stisku tlačítka, nebo na základě kontrolní zprávy, která požaduje odeslání určitého počtu interrupt přenosů. Samotné odeslání je provedeno funkcí `Endpoint_Write_Stream_LE`, které je předán buffer dat, v tomto případě jde o jeden znak s kódem zprávy, a délka bufferu. Před samotným odesláním je nutné funkcí `Endpoint_SelectEndpoint` vybrat koncový bod, na který bude zpráva odeslána. Po odeslání musíme potvrdit odeslání funkcí `Endpoint_ClearIN`.

3.5.7 Bulk přenosy

Hromadné přenosy byly testovány ve třech režimech:

- Odesílání dat hostiteli
- Příjem dat od hostitele
- *Echo*

Před začátkem samotného přenosu je vždy hostitelem zaslána kontrolní zpráva o tom, v jakém režimu má zařízení pracovat. Obvykle je zařízení nastaveno do režimu *echo*.

Po přijetí zprávy o přepnutí zařízení do režimu *odesílání dat* je zavolána funkce `send_bulk`, která nejprve zvolí koncový bod pro hromadné přenosy ve směru k hostiteli a poté v cyklech odesílá bloky dat funkcí `Endpoint_Write_Stream_LE` o celkové velikosti 10MB. Po každém odesláni dat je toto potvrzeno funkcí `Endpoint_ClearIN`.

Po přijetí kontrolní zprávy o přepnutí do režimu *příjmu dat* od hostitele zařízení očekává příchod dat. To je stejně jako u interrupt přenosů zaznamenáno funkcí `checkUSB`, která tentokrát předává řízení funkci `bulk_IO`. Následně je opět vyčtena velikost příchozích dat v zásobníku koncového bodu a tato data jsou načtena funkcí `Endpoint_Read_Stream_LE` a je provedeno potvrzení voláním `Endpoint_ClearOut`. Následně jsou data zkontrolována a v případě neshody je chyba započítána. Po přenesení celého bloku dat si hostitel vyžádá odeslání počtu chyb pomocí kontrolní zprávy.

Režim *echo* funguje podobně jako *přijímání dat* od hostitele, s tím rozdílem, že okamžitě po přijetí bloku dat jsou tato data odeslána zpět hostiteli a mikrokontrolér je nijak nezpracovává.

4 Aplikace hostitelského PC

Hostitelská aplikace byla vyvinuta v prostředí Microsoft Visual Studio 2010 s využitím MFC pro vytvoření uživatelského rozhraní a knihovnou LibUSBx pro implementaci rozhraní USB. Byla použita knihovna LibUSBx ve verzi 1.0.15, vydaná 15. 4. 2013. Aplikace demonstruje využití kontrolních, interrupt a bulk přenosů v synchronním i asynchronním režimu. Isochronní přenosy v současnosti nejsou na platformě Windows podporovány knihovnou LibUSBx, ani LibUSB 1.0, proto nejsou v aplikaci zahrnuty. Isochronní přenosy byly otestovány s využitím starší knihovny LibUSB-Win32, tyto přenosy jsou funkční a v případě potřeby je možné knihovnu LibUSB-Win32 využít.

4.1 Inicializace USB zařízení

Pro správnou funkci každého USB zařízení je třeba po jeho prvním připojení nainstalovat příslušný ovladač. Knihovna LibUSBx dokáže spolupracovat s ovladači WinUSB, libusbK a libusb0. Všechny tyto ovladače byly v rámci testování vyzkoušeny a bez problémů fungují. Pro instalaci ovladače WinUSB je nutné upravit soubor *.inf*, je nutné do něj zapsat Vendor ID a Product ID čísla příslušného zařízení. Ovladače libusbK a libusb0 obsahují podpůrný software, který na základě zadaného Vendor ID a Product ID potřebný ovladač vytvoří, popřípadě jej hned i nainstalují. Všechny tyto ovladače jsou přiloženy na CD.

4.1.1 Inicializace knihovny LibUSBx

Po spuštění aplikace je nutné inicializovat knihovnu LibUSBx, což se provede voláním funkce `libusb_init`, tato funkce musí být zavolána před voláním jakékoliv jiné funkce knihovny LibUSBx. Jejím jediným parametrem je ukazatel na kontext. Využití kontextu umožňuje vícenásobné použití knihovny v rámci jedné aplikace. Pokud se funkci předá hodnota NULL, je využit standardní kontext. V případě úspěšné inicializace vrací funkce 0, v případě neúspěchu vrací chybový kód (viz tabulka 9). Pro získání názvu chyby je možní využít funkci `libusb_error_name`, které se parametrem předá kód chyby a funkce vrací řetězec s jejím názvem.

Knihovna umožňuje využití ladících informací formou zasílání zpráv na standardní výstupy. Voláním funkce `libusb_set_debug` je možné nastavit úroveň podrobnosti zasílaných zpráv. Je možné zachytávat zprávy ve čtyřech úrovních:

- pouze chyby
- chyby a varování
- podrobnější informace
- ladění

Práce s knihovnou LibUSBx se ukončuje voláním funkce `libusb_exit`, které se parametrem předává kontext vzniklý voláním funkce `libusb_init`, případně NULL.

Kód	Název chyby
0	LIBUSB_SUCCESS
-1	LIBUSB_ERROR_IO
-2	LIBUSB_ERROR_INVALID_PARAM
-3	LIBUSB_ERROR_ACCESS
-4	LIBUSB_ERROR_NO_DEVICE
-5	LIBUSB_ERROR_NOT_FOUND
-6	LIBUSB_ERROR_BUSY
-7	LIBUSB_ERROR_TIMEOUT
-8	LIBUSB_ERROR_OVERFLOW
-9	LIBUSB_ERROR_PIPE
-10	LIBUSB_ERROR_INTERRUPTED
-11	LIBUSB_ERROR_NO_MEM
-12	LIBUSB_ERROR_NOT_SUPPORTED
-99	LIBUSB_ERROR_OTHER

Tabulka 9 - Popis chybových kódů knihovny LibUSBx

4.1.2 Načtení seznamu USB zařízení

Po provedení inicializace je potřeba zvolit zařízení, se kterými se bude nadále spolupracovat. K tomu je využita funkce `libusb_get_device_list`, která vrací seznam všech USB zařízení aktuálně připojených k počítači. Seznam načtený funkcí `libusb_get_device_list` musí být po ukončení práce uvolněn funkcí `libusb_free_device_list`.

4.1.3 Otevření zařízení

Otevřením se rozumí získání systémového identifikátor zařízení (handle), díky kterému další funkce k zařízení přistupují.

Knihovna LibUSBx nabízí dvě možnosti otevření připojeného zařízení. První možnost je otevřít zařízení funkcí `libusb_open_device_with_vid_pid`, které se parametrem předá požadované VendorID a ProductID, a jako návratovou hodnotu vrací identifikátor zařízení. Tato možnost je pro tuto práci nepoužitelná, protože umožňuje otevření pouze prvního nalezeného zařízení, ostatní ignoruje.

Využitelná je druhá možnost. Pokud již máme k dispozici seznam všech připojených zařízení, vybereme zařízení, se kterými je aplikace schopná komunikovat. Výběr se provede na základě VendorID a ProductID. Každému připojenému zařízení je odeslán požadavek na zaslání deskriptoru zařízení, který obsahuje VendorID a ProductID. V případě shody je zařízení otevřeno funkcí `libusb_open`. Jako vstupní parametr je předán ukazatel na zařízení z načteného seznamu. V případě úspěšného otevření vrací 0 a výstupním parametrem je předán identifikátor zařízení, v případě neúspěchu funkce vrací chybový kód dle tabulky 15.

4.1.4 Načtení deskriptoru zařízení

Pro každé připojené zařízení je možné vyžádat si jeho deskriptor a to voláním funkce `libusb_get_device_descriptor`, deskriptor obsahuje veškeré nutné informace k identifikaci zařízení. Funkci je parametrem předáván ukazatel na některé zařízení získané funkcí `libusb_get_device_list` a ona výstupním parametrem vrací požadovaný deskriptor.

4.1.5 Načtení deskriptoru konfigurace

Podobně jako deskriptor zařízení můžeme vyčíst deskriptor konfigurace voláním funkce buď `libusb_get_active_config_descriptor`, pro vrácení deskriptoru právě aktivní konfigurace, nebo `libusb_get_config_descriptor` pro deskriptor konfigurace zadané jejím číslem. Načtený deskriptor konfigurace je nutno uvolnit funkcí `libusb_free_config_descriptor`.

4.1.6 Načtení string deskriptorů

String deskriptory obsahují textové informace o zařízení- jeho název, výrobce, případně popis konfigurace nebo rozhraní. Ze zařízení je možné získat string deskriptor použitím funkce `libusb_get_string_descriptor_ascii`. Funkci je parametrem předáno číslo string deskriptoru, který chceme načíst. Funkce vrací výstupním parametrem řetězec daného deskriptoru.

4.1.7 Výběr konfigurace zařízení

Pokud bylo zařízení v pořádku otevřeno, dalším krokem je zvolení konfigurace. Tento krok je důležitý zejména u multifunkčních zařízení, v našem případě volíme jedinou konfiguraci s číslem 0. K výběru konfigurace využíváme funkci `libusb_set_configuration`, které je parametry předán handle zařízení a číslo konfigurace. Pro ověření čísla zvolené konfigurace je možné využít funkci `libusb_get_configuration`, která ve výstupním parametru předává číslo aktuálně zvolené konfigurace. Obě tyto funkce předávají jako návratovou hodnotu kód dle tabulky 15.

4.1.8 Připojení k rozhraní

Podobně jako výběr konfigurace funguje připojení k rozhraní. Aby bylo možné využívat komunikaci adresovanou koncovým bodům, je nutné provést volání funkce `libusb_claim_interface`, které se předává parametry handle zařízení a číslo zvolené konfigurace. Po dokončení práce s daným rozhraním by měla být zavolána funkce `libusb_release_interface`, která vrací zařízení do jeho standardního nastavení.

Pokud všechny předešlé kroky proběhly bez chyby, je dané zařízení připraveno ke komunikaci a přidáno do seznamu připojených zařízení.

4.2 Kontrolní přenosy

Kontrolní přenosy jsou vždy adresovány kontrolnímu koncovému bodu 0 a slouží především k řízení zařízení. Uživatelsky definované kontrolní zprávy je možné odesílat pomocí funkce `libusb_control_transfer`, směr přenosu dat je v případě těchto zpráv určen parametrem `bmRequestType`, který musí obsahovat jednu z hodnot výčtu `libusb_endpoint_direction` buď `LIBUSB_ENDPOINT_IN`, nebo `LIBUSB_ENDPOINT_OUT`. Prototyp funkce:

```
int libusb_control_transfer (
	libusb_device_handle * dev_handle,	// identifikator zarizeni
	uint8_t bmRequestType,			// typ zpravy (smer, typ, cil)
	uint8_t bRequest,			// kod zpravy
	uint16_t wValue,			// hodnota- datove pole
	uint16_t wIndex,			// index- datove pole
	unsigned char * data,			// data k odeslani/prijata data
	uint16_t wLength,			// velikost dat k odeslani /
						// maximalni velikost prijatych dat
	unsigned int timeout			// cas, po ktery funkce ceka na odpoved
	);
```

Příklad použití (načtení počtu chyb přenosu):

```
	int errors;
	int result = libusb_control_transfer(handle, LIBUSB_REQUEST_TYPE_VENDOR |
		LIBUSB_RECIPIENT_DEVICE | LIBUSB_ENDPOINT_IN, REQ_GET_ERRORS, 0, 0,
		(unsigned char *)&errors, sizeof(errors), 2000);
```

4.3 Synchronní přenosy

Synchronní přenosy jsou jednodušší variantou pro USB komunikaci. Tyto přenosy pracují v blokujícím režimu, což znamená, že funkce pozastaví běh programu do okamžiku jejího dokončení, popřípadě do vypršení nastaveného času. V synchronním režimu mohou pracovat kontrolní, bulk a interrupt přenosy. Pro bulk přenosy využíváme funkci `libusb_bulk_transfer` a pro interrupt přenosy `libusb_interrupt_transfer`. Tyto dvě funkce mají shodné parametry:

Název parametru	Popis
dev_handle	handle zařízení
endpoint	adresa koncového bodu
data	data k odeslání, nebo přijatá data
length	velikost dat k odeslání, nebo maximální velikost přijatých dat
transferred	výstupní parametr- velikost aktuálně přenesených dat
timeout	čas, po který funkce čeká na odpověď

Tabulka 10- Parametry synchronních přenosů

Příklad použití (bulk odeslání):

```
int i, result;
result = libusb_bulk_transfer(handle, EP_BULK_OUT, msg,
       strlen((char *)msg) + 1, &i, 1000);
```

Příklad použití (bulk přijetí):

```
int i, result;
result = libusb_bulk_transfer(handle, EP_BULK_IN, msg,
       EP_BULK_SIZE, &i, 1000);
```

4.4 Asynchronní přenosy

Asynchronní přenosy jsou silnější a komplexnější variantou USB přenosů, nabízejí vyšší výkon a více možností využití, než synchronní přenosy. Všechny typy USB přenosů je možné používat asynchronně. Asynchronní přenos je možné popsat v pěti krocích:

1. Alokace přenosu
2. Naplnění instance přenosu informacemi
3. Zpracování přenosu
4. Dokončení a zpracování přenosu
5. Uvolnění přenosu

Hlavní rozdíl oproti synchronním přenosům je oddělení zpracování a dokončení přenosu. V programu je nejprve zavolána neblokující funkce, která přenos spustí, a po dokončení přenosu je aplikace o této skutečnosti informována vyvoláním callback funkce.

Prvním krokem k zahájení asynchronního přenosu je **alokace**. Ta se provádí voláním funkce `libusb_alloc_transfer`, jejím jediným parametrem je počet isochronních paketů, které mají být alokovány, při využití ostatních přenosů se tento parametr nastavuje na 0. Funkce vrací ukazatel na alokovanou instanci struktury `libusb_transfer`.

V dalším kroku se provede **naplnění** této instance informacemi o přenosu. Jsou dvě možnosti jak toto provést- ručně naplnit instanci `libusb_transfer`, nebo využít připravené funkce, pro bulk přenosy `libusb_fill_bulk_transfer`, pro interupt `libusb_fill_interrupt_transfer`. Tyto funkce naplní instanci `libusb_transfer` informacemi o přenášených datech, adrese koncového bodu, časovém limitu pro přenos a callback funkci, která bude zavolána po dokončení přenosu.

libusb_transfer	
Název	Popis
dev_handle	handle zařízení
flags	příznaky
endpoint	adresa koncového bodu
type	typ přenosu
timeout	časový limit pro přenos
status	stav přenosu
length	velikost bufferu
actual_length	velikost aktuálně přenesených dat
callback	callback funkce pro zpracování přenosu
user_data	uživatelská data pro zpracování
buffer	datový buffer
num_iso_packets	počet isochronních paketů
iso_packet_desc[]	deskriptory isochronních přenosů

Tabulka 11 - Popis struktury libusb_transfer

Třetím krokem je samotné **zpracování** přenosu, což znamená vytvoření roury mezi hostitelem a daným koncovým bodem a spuštění naslouchání, nebo odesílání dat. Zpracování se provádí voláním funkce `libusb_submit_transfer`, která má jako jediný parametr instanci alokované a naplněné struktury `libusb_transfer`. Funkce v případě úspěchu vrací 0, jinak příslušný chybový kód.

O **dokončení** přenosu je aplikace informována vyvoláním callback funkce, která byla vložena do struktury `libusb_transfer`. O stavu, v jakém byl přenos dokončen, informuje hodnota položky status ze struktury `libusb_transfer`.

enum libusb_transfer_status	
Hodnota	Popis
LIBUSB_TRANSFER_COMPLETED	Přenos úspěšně dokončen
LIBUSB_TRANSFER_ERROR	Chyba při přenosu
LIBUSB_TRANSFER_TIMED_OUT	Vypršel časový limit přenosu
LIBUSB_TRANSFER_CANCELLED	Přenos byl zrušen
LIBUSB_TRANSFER_STALL	Bulk/interrup: koncový bod pozastaven Control: požadavek není podporován
LIBUSB_TRANSFER_NO_DEVICE	Zařízení bylo odpojeno
LIBUSB_TRANSFER_OVERFLOW	Přijato více dat, než bylo očekáváno

Tabulka 12 - Přehled stavů dokončení přenosu

Po dokončení práce s přenosy, je nutné každý alokovaný transfer **uvolnit** voláním funkce `libusb_free_transfer`.

4.4.1 Asynchronní interrupt přenosy

Asynchronních interrupt přenosů je v aplikaci využito pro zachycení událostí vzniklých na straně zařízení. Alokace, naplnění a spuštění je provedeno hned při vytvoření zařízení v aplikaci voláním funkce `start_int_listen`. Příklad spuštění přenosu:

```
struct libusb_transfer * transferIN = NULL;
transferIN = libusb_alloc_transfer(0);
libusb_fill_interrupt_transfer(transferIN, handle,
     EP_INT_IN, &buffer, 1, (libusb_transfer_cb_fn)
     handle_event, NULL, 10000);
libusb_submit_transfer(transferIN);
```

K zachycení události vzniklé na USB zařízení slouží funkce `libusb_handle_events`. Tato funkce vyvolá callback funkci, v okamžiku dokončení přenosu. Tato funkce pracuje v blokujícím režimu. Musí být volána buď po každém volání `libusb_submit_transfer`, nebo je možné tuto funkci volat v jiném vlákně. Volání v jiném vlákně neblokuje aplikaci a programátor nemusí hlídat její volání. Při dokončení přenosu ve stavu LIBUSB_TRANSFER_TIMED_OUT je vyvolána funkce `restart_int_listen` , která obnovuje přenos voláním `libusb_submit_transfer`. V případě dokončení ve stavu LIBUSB_TRANSFER_COMPLETED, je vyhodnocen přijatý buffer a přičtena událost.

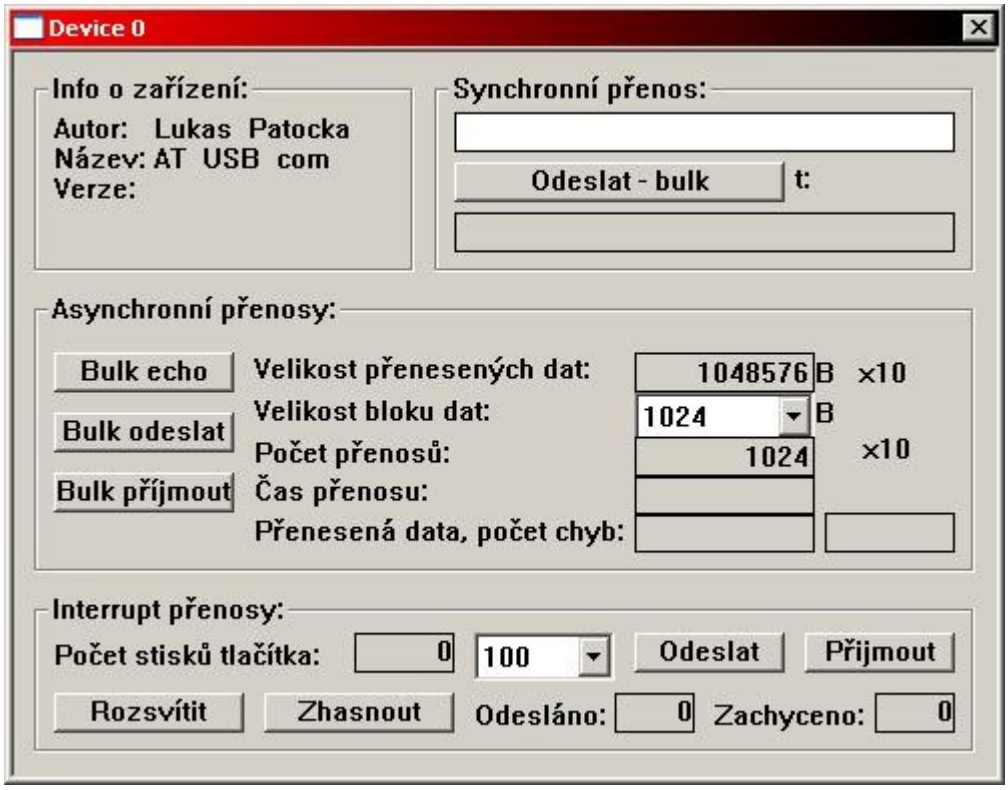
4.4.2 Asynchronní bulk přenosy

V aplikaci jsou asynchronní bulk přenosy spouštěny na základě uživatelského vstupu. Nyní popíši pouze přenos v režimu *echo*, protože zahrnuje oba další přenosy. Nejprve je nutné alokovat a naplnit dva přenosy, jeden pro směr od hostitele, druhý pro směr od zařízení. Následně je v cyklu volána funkce `libusb_submit_transfer`, nejprve pro odeslání, následně pro příjem. K zachycení dokončení přenosu i v tomto případě slouží stejná callback funkce jako pro interrupt přenosy. To je možné díky předání instance struktury `libusb_transfer` do callback funkce, takže je možné odlišit typ přenosu i koncový bod kterému byl adresován. V případě bulk přenosů callback funkce pouze zajišťuje vyčkání dokončení jednoho přenosu, aby mohl začít další. Vyhodnocení času přenosu a chybovosti probíhá až po dokončení přenosu celého 1MB. Ukázka komunikace v režimu *echo*:

```
struct libusb_transfer *transferIN = NULL;
struct libusb_transfer *transferOUT = NULL;
//     alokace transferu
transferOUT = libusb_alloc_transfer(0);
transferIN = libusb_alloc_transfer(0);
//     naplneni transferu informacemi
libusb_fill_bulk_transfer(transferOUT, handle, EP_BULK_OUT, usb->buffer,
       bufferSize, (libusb_transfer_cb_fn)handle_event, ptrOUT, 10000);
libusb_fill_bulk_transfer( transferIN, handle, EP_BULK_IN, bufferIN,
       bufferSize, (libusb_transfer_cb_fn)handle_event, ptrIN, 10000);
//     cyklus odesilani a prijimani dat
for(int i = 0; i < count; i++)
{
       libusb_submit_transfer(transferOUT);
       libusb_submit_transfer(transferIN);
}
libusb_free_transfer(transferIN);
libusb_free_transfer(transferOUT);
```

4.5 Popis hostitelské aplikace

Úvodní okno aplikace obsahuje pouze seznam nalezených USB zařízení a textový výstup s informacemi o chodu aplikace. Hlavní okno zařízení je popsáno dále:



Obrázek 4 - Rozhraní aplikace

Informace o zařízení

V této kolonce jsou zobrazeny informace získané ze zařízení pomocí string deskriptorů. Načtení string deskriptorů je popsáno v kapitole 4.1.6.

Synchronní přenos

Zde je možné otestovat si jednoduchý synchronní bulk přenos v režimu *echo*. Do horního vstupního pole je možné napsat jakýkoliv text, který je po stisku tlačítka odeslán, následné přijat a zobrazen ve spodním výstupním poli.

Asynchronní přenosy

Tato část aplikace je věnována asynchronním přenosům. Umožňuje otestování hromadného odeslání, přijetí a režimu *echo*. V pravé části je zobrazena celková velikost přenášených dat, je možné zvolit velikost bloku dat, po které budou data zpracovávána (od 64B po 0,5MB), dále je zobrazen počet uskutečněných přenosů, po dokončení čas přenosu, velikost přenesených dat a počet chyb. Tlačítky „Bulk echo“, „Bulk odeslat“ a „Bulk přijmout“ jsou přenosy odstartovány. Každý přenos proběhne desetkrát po sobě. Aplikace ukládá výsledky přenosů do souborů, pro každý typ přenosu a velikost bloku dat zvlášť.

Interrupt přenosy

V levé části tohoto pole je možné otestovat okamžitou odezvu mezi zařízením a hostitelem. Položka „Počet stisků tlačítka“ slouží k demonstraci přenosu ze strany zařízení- po každém stisku tlačítka na desce je hodnota zvýšena. Tlačítka „Rozsvítit“ a „Zhasnout“ slouží k ovládání LED diody ze strany hostitele. V pravé části je možné otestovat spolehlivost doručení interrupt zpráv, je zde možnost vybrat počet zpráv, které budou odeslány, buď ze strany hostitele, nebo ze strany zařízení. Počet přijatých zpráv je zobrazen v poli „Zachyceno“.

4.6 Měření

Podmínky pro měření:

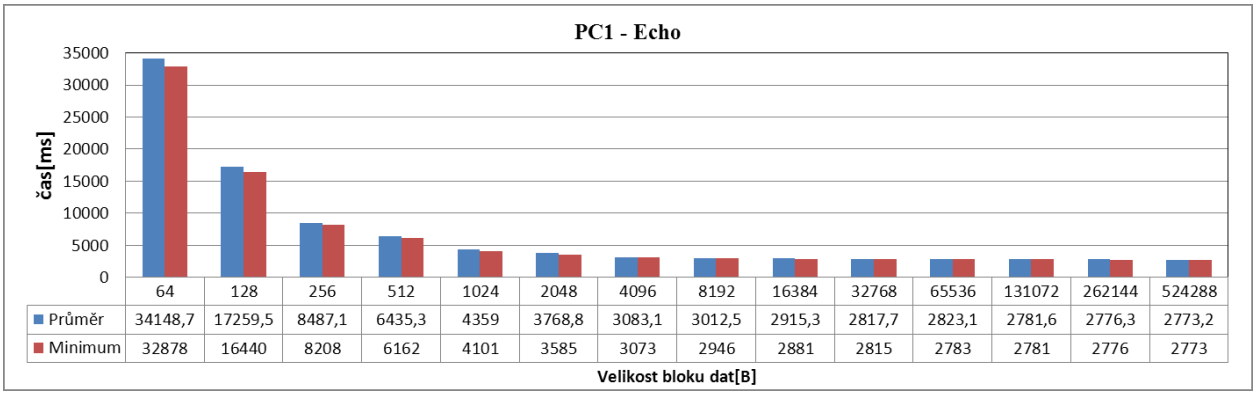
Měření bylo provedeno na dvou počítačích:

PC1: Notebook Toshiba
Pentium Dual-Core CPU T4400 @ 2,20GHz, 2,96GB RAM
Microsoft Windows XP Profesional, Service Pack 3

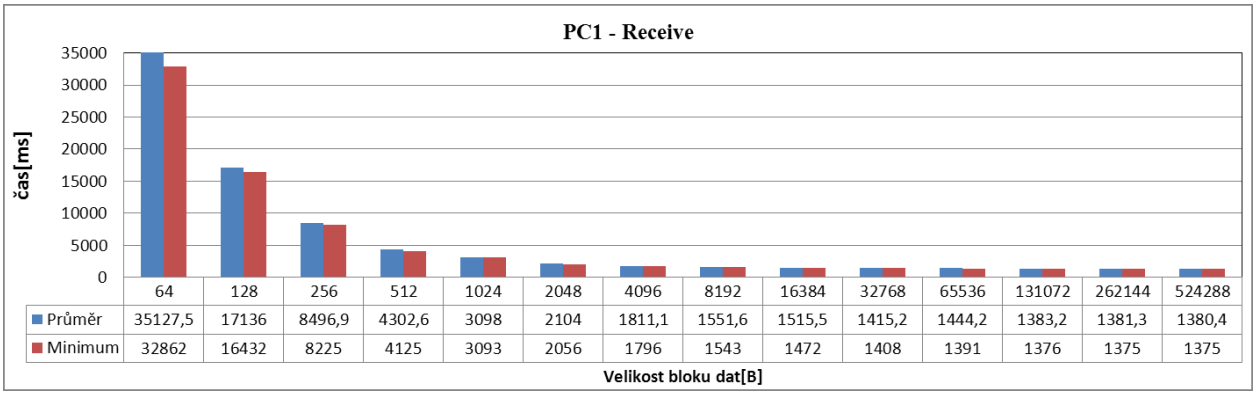
PC2: PC
AMD Athlon 64 X2 Dual-Core 5000+ 2,60GHz, 2GB RAM
Microsoft Windows 7 Home Basic 64bit, Service Pack 1

4.6.1 Měření rychlosti a chybovosti bulk přenosů

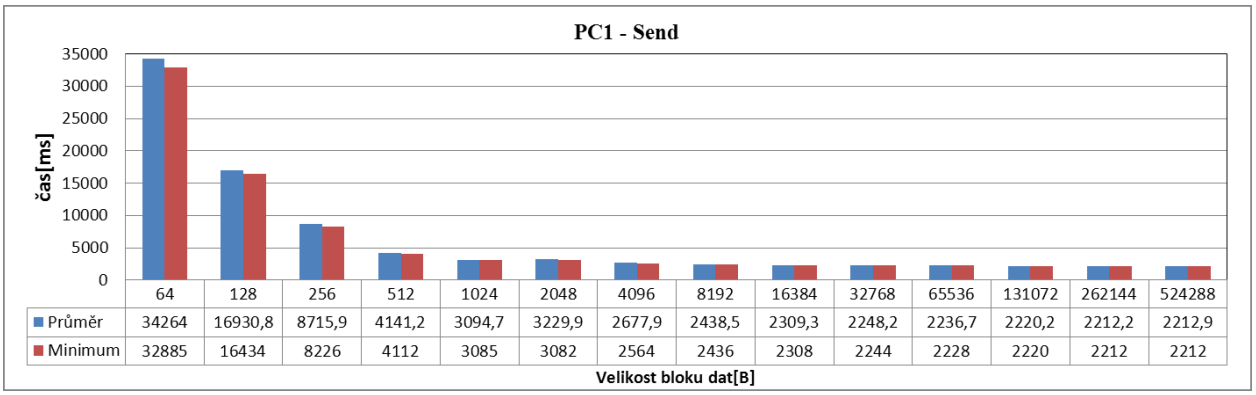
Měření Bulk přenosů bylo provedeno v režimech odesílání dat ze strany počítače, odesílání dat ze strany mikrokontroléru a *echo*- odeslání ze strany počítače a zpětné odeslání ze strany mikrokontroléru. Pro každý z těchto režimů bylo provedeno 14 měření s různou délkou přenášených bloků dat od 64B po 0,5 MB. V každém měření byl přenesen 1MB dat desetkrát po sobě. V grafech jsou zaznamenány průměrné a minimální hodnoty z těchto deseti měření.



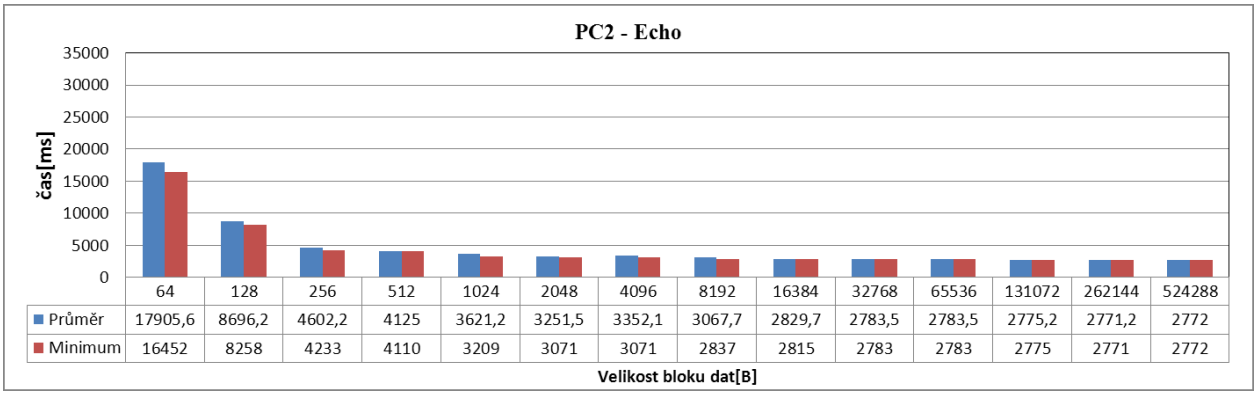
Graf 1 - PC1 *Echo*



Graf 2 - PC1 Příjem dat

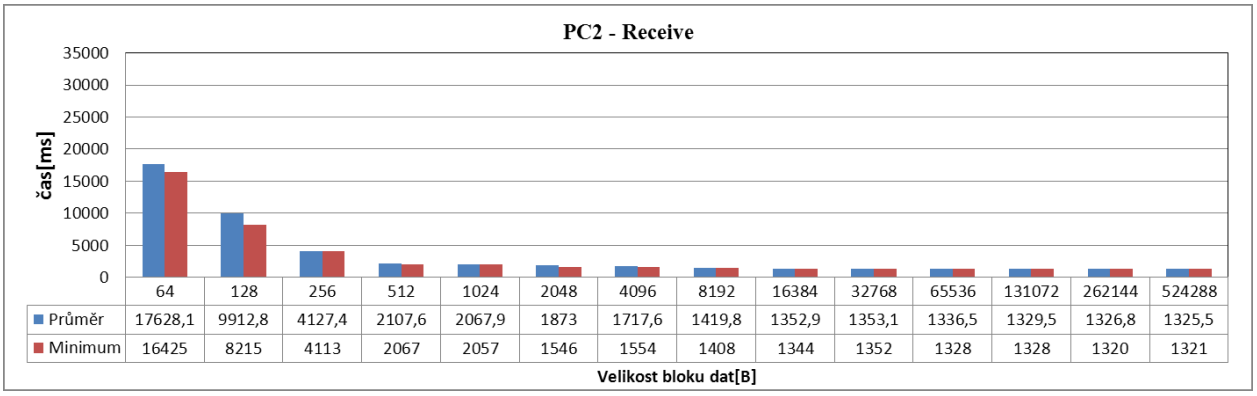


Graf 3 - PC1 Odesílání dat



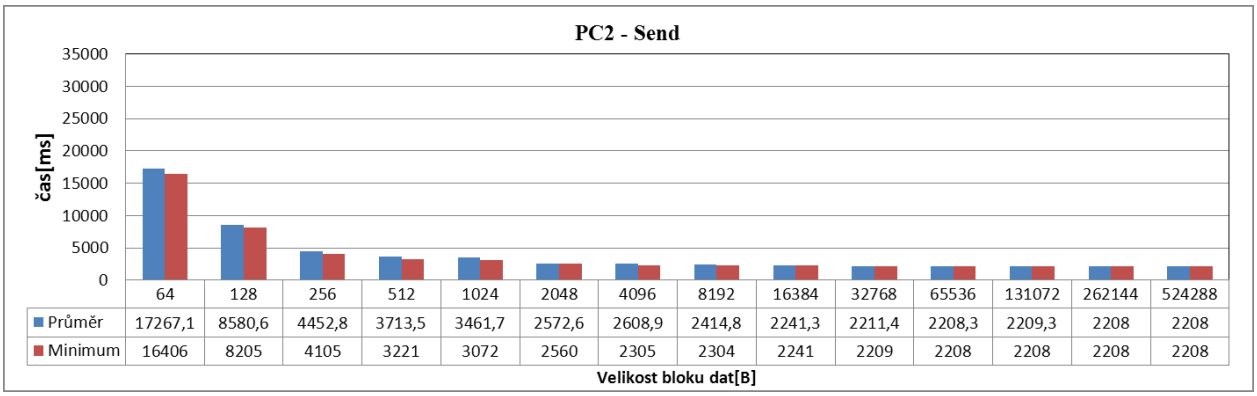
	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288
Průměr	17905,6	8696,2	4602,2	4125	3621,2	3251,5	3352,1	3067,7	2829,7	2783,5	2783,5	2775,2	2771,2	2772
Minimum	16452	8258	4233	4110	3209	3071	3071	2837	2815	2783	2783	2775	2771	2772

Graf 4 - PC2 *Echo*



	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288
Průměr	17628,1	9912,8	4127,4	2107,6	2067,9	1873	1717,6	1419,8	1352,9	1353,1	1336,5	1329,5	1326,8	1325,5
Minimum	16425	8215	4113	2067	2057	1546	1554	1408	1344	1352	1328	1328	1320	1321

Graf 5 - PC2 Příjem dat



	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288
Průměr	17267,1	8580,6	4452,8	3713,5	3461,7	2572,6	2608,9	2414,8	2241,3	2211,4	2208,3	2209,3	2208	2208
Minimum	16406	8205	4105	3221	3072	2560	2305	2304	2241	2209	2208	2208	2208	2208

Graf 6 - PC2 Odesílání dat

5 Závěr

S využitím knihovny LibUSBx, mikrokontrolérem Atmel AVR AT90USB1287, podporovaným knihovnou LUFA, se podařilo vytvořit funkční aplikaci pro ověření USB komunikace. Hostitelská aplikace je schopna rozpoznat a obsloužit více zařízení. Byly otestovány všechny typy přenosů, pouze izochronní přenosy byly testovány s využitím starší knihovny LibUSB Win32 a nebylo s nimi provedeno měření, kvůli tomu že knihovna LibUSB Win32 nepodporuje asynchronní režim přenosů.

5.1 Zhodnocení naměřených výsledků

Bulk přenosy:

Při testování bulk přenosů nebyla zaznamenána jediná chyba, ani žádná ztracená data. V rámci testování bylo přeneseno několik desítek až stovek MB dat. Vzhledem k těmto skutečnostem je možné prohlásit bulk přenosy za spolehlivé a bezchybné. V době přenosu byly zjištěny značné rozdíly vzniklé různou velikostí přenášených bloků dat. Malé bloky dat byly přenášeny zásadně pomaleji (pro blok 64B byla na PC2 naměřena přenosová rychlost přibližně 62kB/s) než velké bloky (pro blok 512kB byla naměřena přenosová rychlost přibližně 775kB/s). Z uvedených grafů je patrné, že přenosovou rychlost značným způsobem ovlivňuje i použitá sestava hostitelského počítače. Rozdíly jsou patrné především na malých blocích dat, kde je čas přenosu na PC2 téměř poloviční oproti PC1, ale na přenosech po velkých blocích dat je rychlost téměř shodná. Rozdíl může být způsoben rozdílnou architekturou mikroprocesoru, nebo použitím jiného operačního systému. USB standard pro režim full speed uvádí maximální přenosovou rychlost 12Mb/s, což je 1,5MB/s, této rychlosti při měření bulk přenosů nebylo dosaženo. Větší přenosové rychlosti by mohlo být dosaženo pomocí izochronních přenosů, protože neobsahují kontrolu dat a případné opakování paketů.

Interrupt přenosy:

V případě interrupt přenosů není vhodné měřit rychlost, protože ta by měla být z principu těchto přenosů konstantní a nepředpokládá se u nich přenos většího množství dat. Proto bylo provedeno pouze měření spolehlivosti doručení interrupt zpráv. Bylo provedeno 10 měření po 1000 zprávách v obou směrech přenosu a vždy byly doručeny všechny zprávy. Dá se tedy říci, že spolehlivost interrupt přenosů je 100%.

6 Zdroje:

[1] LibUSB [online]. [16. 2. 2013].
< http://libusb.org/ >

[2] LibUSB wiki: APIs [online]. [16. 2. 2013].
< http://libusb.org/wiki/APIs/ >

[3] LibUSB wiki: libusb-win32 [online]. [16. 2. 2013].
< http://libusb.org/wiki/libusb-win32 >

[4] LibUSB wiki: windows_backend [online]. [16. 2. 2013].
< http://libusb.org/wiki/windows_backend >

[5] GitHub – LibUSBx [online]. [17. 2. 2013].
< https://github.com/libusbx/libusbx/wiki >

[6] GitHub – LibUSBx: milestones [online]. [17. 2. 2013].
< https://github.com/libusbx/libusbx/issues/milestones >

[7] Atmel AVR 8-bit and 32-bit Microcontroller [online].
< http://www.atmel.com/products/microcontrollers/avr/default.aspx >

[8] Atmel AT90USB1287 Datasheet [revision L, updated: 08/2012].
< http://www.atmel.com/Images/7593S.pdf >

[9] LUFA Library [online]. [30. 3. 2013].
< http://www.fourwalledcubicle.com/LUFA.php >

[10] LUFA vs the Atmel 8-bit USB AVR Stack [online].
< http://www.fourwalledcubicle.com/files/LUFA/Doc/120219/html/_page__l_u_f_avs_atmel_stack.html >

[11] USB.org USB 2.0 Specification [Revision 2.0] .
< http://www.usb.org/developers/docs/usb_20_040413.zip >

7 Seznam tabulek

Tabulka 1 - Přehled mikrokontrolérů řady AT90USB 10
Tabulka 2 - Přehled USB standardů 12
Tabulka 3 - Popis deskriptoru konfigurace 18
Tabulka 4 - Popis deskriptoru rozhraní 18
Tabulka 5 - Popis deskriptorů koncových bodů 19
Tabulka 6 - Hodnoty konstant koncových bodů 20
Tabulka 7 - Popis uživatelsky definovaných kontrolních zpráv 21
Tabulka 8 - Seznam použitých kontrolních zpráv 22
Tabulka 9 - Popis chybových kódů knihovny LibUSBx 27
Tabulka 10- Parametry synchronních přenosů 29
Tabulka 11 - Popis struktury libusb_transfer 31
Tabulka 12 - Přehled stavů dokončení přenosu 31

8 Seznam obrázků

Obrázek 1 - Deska plošného spoje 11
Obrázek 2 - schéma zapojení 11
Obrázek 3- Hierarchie USB zařízení 13
Obrázek 4 - Rozhraní aplikace 33

Graf 1 - PC1 *Echo* 36
Graf 2 - PC1 Příjem dat 36
Graf 3 - PC1 Odesílání dat 36
Graf 4 - PC2 *Echo* 37
Graf 5 - PC2 Příjem dat 37
Graf 6 - PC2 Odesílání dat 37

9 Obsah CD

App_MC		- aplikace mikrokontroléru
	Code	- složka projektu
	HEX	
	AT_USB_com.hex	- binární soubor programu mikrokontroléru
App_PC		
	Code	- složky projektů
	EXE	
	USB_Control.exe	- hostitelská aplikace
	USB_test.exe	- testovací aplikace USB zařízení
Library		
	LUFA	- knihovna LUFA
	LibUSBx	- knihovna LibUSBx
Drivers		
	WinUSB	- ovladač WinUSB
	libusbK-3.0.5.16-setup.exe	- ovladač libusbK (libusb0)